Welcome!

CSCI(41);

(There's a reason I'm not making the presentation full screen!)

I will give you all my slides & code on Canvas

Outline

- Who is this dude who's talking to me?
- Logistical stuff
- What this class is about
- Fancy Object-Oriented Programming (OOP) in C++

About Me

Lawton Nichols

Mr.

Dr.

Professor

- He/him
- Please call me Lawton
- Went to Fresno State for undergrad, UC Santa Barbara for grad school
  - All three of my degrees are in Computer Science
- I have the cutest cat! His name is Lonzo.
  - Named after Alonzo Church, Ph.D. advisor of Alan Turing

Thank you in advance
for putting up with me <3

Info & Logistics

I’m gonna move quickly here—remember to read the syllabus

Asynchronicity

- I'll record every lecture
- I'll give you a ~24-hour window to start/take exams
  - The exams themselves will be timed once you start them
    - e.g.: 1h15m for a midterm
- "Attendance" is your Check-In Surveys—if you miss 2 in a row (or the very first one), I will drop you

Setting Expectations

- I’ve been told that this is the hardest CSCI class
- A lot of traditionally difficult concepts to teach/learn
  - So. Many. Pointers. Oof
- The concepts can be a little dry compared to CSCI 40. I’ll try to make them as exciting as possible!
- Expect to write about 80–100ish lines of code each week
- I’ll be here to help you as much as I can!

Technology Loan Program

- If you need a laptop or a better internet connection, see the Resources page on Canvas!

The Syllabus

- Homework and Programming Assignments (a.k.a. labs) pretty much every week
- Check-In Surveys: me forcing you to psychoanalyze yourself
- Two midterms
  - Week 6, Week 12
- Grade breakdown
- There will be extra credit
  - 0.5% extra credit for going to any ACM Club workshop
  - You can only get a maximum of 10% extra credit
- Final exam date

Dropping Lowest Scores

- I will drop:
  - Your lowest 3 labs
  - Your lowest 3 zyBooks reading assignments (homeworks)

Due Dates

- All assignments in this class (except the check-in surveys) **will be due at 5pm**
  - This ensures that I'm awake when inevitable pre-/post-due-date issues occur
- **Equity exceptions:**
  - If you work ≥ 20 hours per week (school doesn't count), I'll extend your due dates an extra **7 hours**
  - If you work ≥ 40 hours per week, care for a child, etc., I'll extend your due dates an extra **15 hours**
  - Contact me if either of these apply to you
    - These are for **assignments only**, not exams

Late Submissions

- I will accept late zyBooks assignments, labs, and check-in surveys **up to the last day of instruction**
  - 50% penalty
  - Please remind me if I don’t re-grade your late zyBooks or check-in survey submissions within a week or so. The correct penalty is applied to labs automatically.

zyBooks

- Our “textbook” is virtual and interactive, and I’ll assign you homework out of it to complete. You must buy it.
  - The interactivity will seriously help you learn the material, so I think it’s worth it
- You can buy it from the bookstore or directly from the zyBooks
  - **Check the prices/fine print!!!** One way might be cheaper, easier to return, etc.
  - **DO NOT BUY IT BY GOING TO THE ZYBOOKS WEBSITE YOURSELF**
    - Follow the link in the first assignment
- All the assignments are up, and I encourage you to read ahead if you feel like it
- Let’s go look at it!

Discord

- Demo!
- I love you, but I won't accept your Discord friend request <3
- I encourage you to **ask questions publicly** when you can (so that your peers can also help)
  - **Don't** share answers or lots of code publicly, though
- You can always privately ask me questions via a direct message (DM)
- I ***strongly*** encourage you to use Discord to contact me instead of email or Canvas

- *(I will make every effort to respond within 24 hours on weekdays, and 48 hours on weekends. I like to pretend that I have a life, so I might be "online" but not on our server.)*

Come to Office Hours!

- On Zoom
- Check Canvas for the schedule and Zoom link
- You can also make an appointment

You will get stuck. Ask for help!!!

- If you've been stuck for ~30 minutes on a problem and can't figure out what to do, please ask for help!
- It's *my job* to get you over your current hurdle and on to the next one
- That being said, getting stuck for shorter periods of time is expected

So what's CSCI 41 about?

Algorithm Analysis and Data Structures!

- We have a ***NEED FOR SPEED*** in this class
  - We don't just care about getting the job done anymore
- We want better ways to structure and manage our data
  - What is a program but a glorified data-shuffler?

But first, we need to learn to drive C++ better

- We are going to go deep into C++ classes and **object-oriented programming**
- We'll first (optionally) review:
  - pointers, arrays, vectors
  - new/delete
  - const
  - references
  - exceptions
- Watch the review video if you need to

Our Beautiful Class Website

- Go to www.lawtonsclass.com
- Your username **and** password right now are both:
  - (your last name, all lowercase) + _ + (your student id)
  - Examples:
    - Nichols, 08675309 → nichols_08675309
    - García-Márquez, 01234567 → garciamarquez_01234567
- Let me know if you're enrolled in the class but can't log in—everyone else just hang out for a few minutes

And now for the actual lecture

Intro to OOP

csci(41);

I'm gonna throw a lot of stuff at you right now

A great way to study is to try all this code out yourself!

What I want you to be able to do

- Explain the difference between a struct and a class
- Analyze the completeness of a set of test cases
- Classify the running time of functions as faster or slower than others

Peer Instruction & In Class Exercises

- I will ask questions for you to answer using the byeClicker app
- I'll sometimes split you into pairs so that you can help each other learn—this is called **peer instruction**
  - I promise it's worth the slight discomfort
- Our first question is on the next slide!
- It's an in-class exercise (ICE), and our mascot here is called the Crush, so naturally I'm going to call them…forgive me…

Crushed ICE

Answer the following question on byeClicker after I press start:

- What is your major?
  - (a) Computer Science
  - (b) Computer Engineering
  - (c) Other Engineering
  - (d) Other STEM
  - (e) Something else

Crushed ICE

- What is your familiarity/confidence with using the UNIX/Linux terminal?
  - (a) Know nothing or almost nothing about it
  - (b) Used it a little; beginner level
  - (c) Some expertise; lots of gaps though
  - (d) Lots of expertise; a few gaps
  - (e) Know too much; I have no life

Crushed ICE

- Have you ever programmed your own data structures before?
  - (a) No, never
  - (b) Yes, but it's been a while
  - (c) Yes, recently
  - (d) I could teach this class if I wanted to

Namespaces, header files, and implementation files

What's in a name?

using namespace std;

- A **namespace** keeps separate things with the same name separate
- Now you have options:
  - **cout** << foo::f(5) << **endl**;
  - **cout** << bar::f(5) << **endl**;
  - And if for some reason you decide to write your own **cout**, you can always access the standard library's version through **std::cout**
- You already know how to import all of a namespace with the **using** keyword

```
namespace foo {
  void f(int x) {
    return x+1;
  }
}

namespace bar {
  void f(int x) {
    return x+2;
  }
}
```

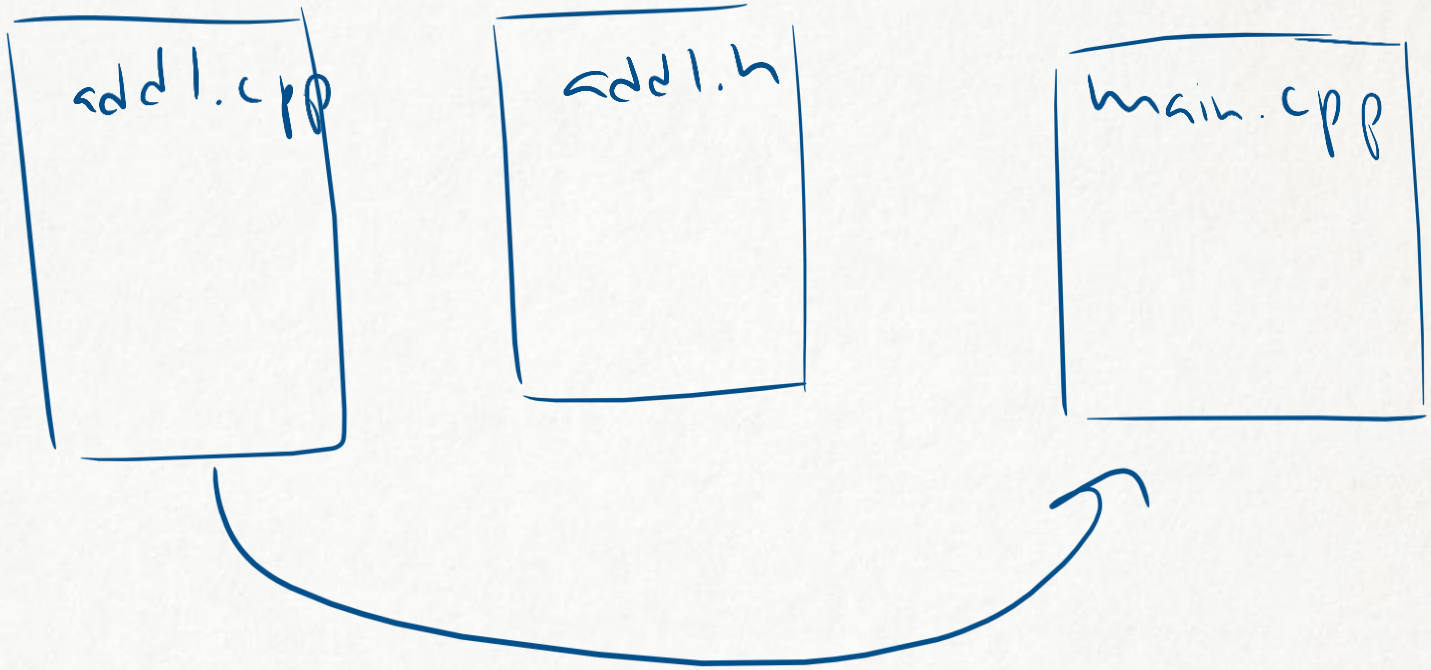
:: ≡ 's

Implementation and Header Files

add1: the world's most complicated function

- It's a good idea to use more than one file to write your programs
- Implementation files: .cpp/.cc/.cxx
  - Contain the actual implementations (i.e., function bodies)
- Header files: .h/.hpp/.hh/.hxx
  - Contain **declarations** so that you can use the corresponding .cpp/.cc/.cxx file *as a library*
  - Function declarations *and* class declarations
  - *Never use a namespace in a header file*
- #ifndef FILENAME, #define FILENAME, #endif
  - **Header file guards** to keep *infinite inclusion* from happening
- add1.cpp, add1.h, main.cpp

add1 library



GNU C++ compiler:
g++ file1.cpp file2.cpp ... -o output_exe

You only have to give the implementation files, because `#include "add1.h"` copies and pastes the file into the current file.

Structs

Let's make a program that keeps track of the following information as a user enters it

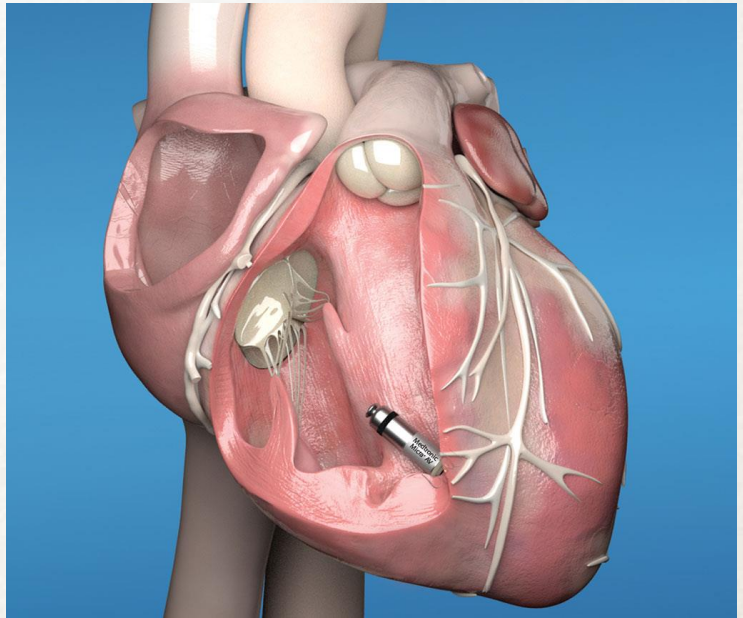
Vars in main:

- **int** count;
- **int** currentMax; ← *Continuously updated!*

(Stop when the user enters -1)

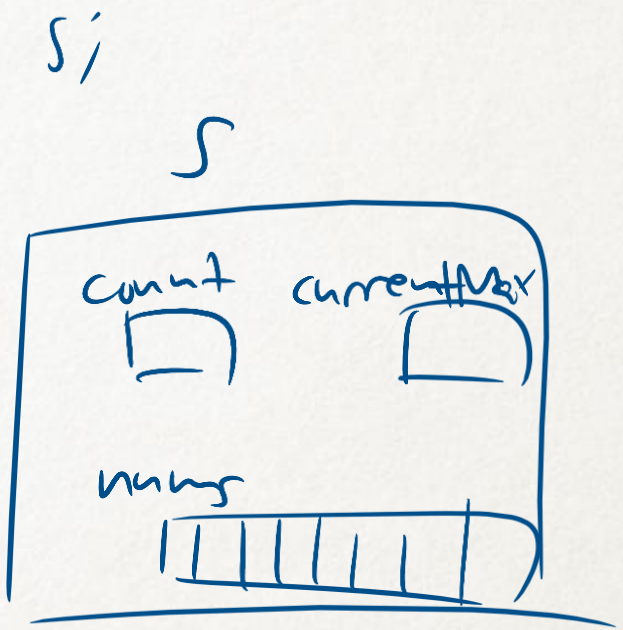
g++ pacemaker.cpp –o pacemaker

Perhaps this is part of a pacemaker, keeping track of recent heart rates



This information should be grouped together in one place. Let's use a struct!

- Sometimes called a "plain old data structure" (POD)
- A struct holds a bunch of different types at the same time
- We'll make a Stats struct that holds:
  - **int** count;
  - **int** currentMax;
  - **vector<int>** nums;



Classes, member functions, and friends

It'd be nice to have some operations ***associated*** *with the struct rather than hanging around outside.*

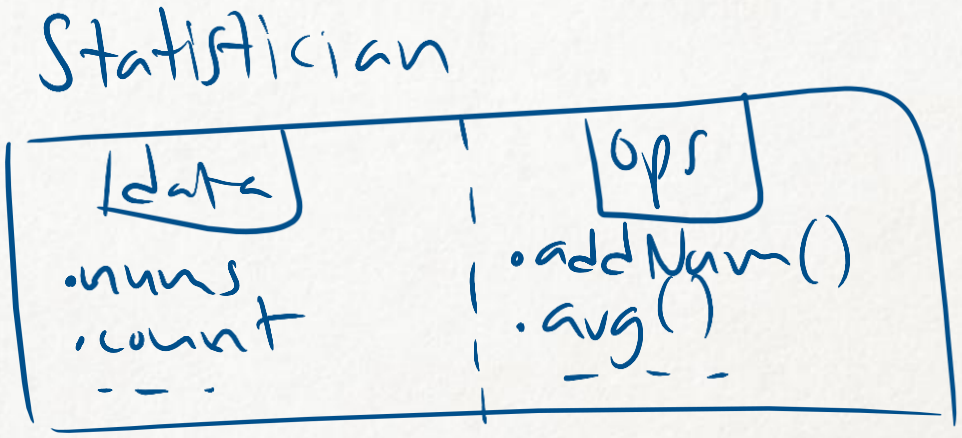
Time for classes! We'll call ours...

The Statistician

Operations: **addNumber**, **getMax**, **average**

We'll also need an **init** operation to give our member vars initial values.
(42 for the currentMax, {42} for nums, 1 for count)

(Everything public for now)



Crushed ICE

- I'm going to split you up into peer instruction groups!
  - Take a minute to introduce yourself and a minute or two to solve the problem below
- We'd like to order our methods from *fastest* to *slowest*. Which ordering do you think is correct?

(a) init, addNumber, getMax, average
(b) init, getMax, addNumber, average
(c) getMax, init, addNumber, average
(d) getMax, init, average, addNumber
(e) Something else / It depends

How "fast" are these member functions, relatively?

- init:

  3 lines, not too bad

- addNumber:

  a few lines / conditionals

- getMax:

  def the fastest!

- average:

  depends on the length of the vec

public* and *private

- Why does our current class have a problem?
- We don’t want to be able to modify any of our member variables outside of our member functions.
- We know what’s best for our class, not somebody else!

frens

- Let’s make an add1ToEveryNum function *outside of the class*. How ever shall we access the private data that we need?!
- The **friend** keyword allows a class to give access to its private member variables.
- The class gets to decide who sees the private member vars.

Testing

How in the world do we know that our Statistician implementation is correct?

We have to make a bunch of **tests**!

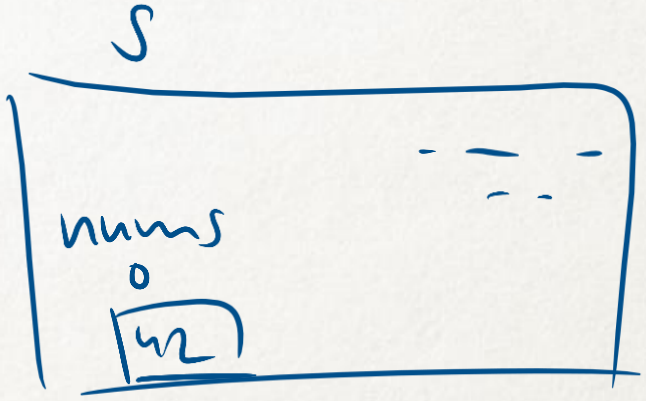
Tests help to convince ourselves that our program is correct.

A test runs a given piece of code with a **predefined input** for which we know what the correct output should be.

I know that the following code:

```
Statistician s;
s.init();
```

Results in s.nums.at(0) == 42, etc.



Example: testing the getMax method with assertTrue

- Our test case:
  - Make a new Statistician s
  - Call init() on it
  - getMax() should return 42 right now
  - Add 43 to s
  - getMax() should return 43 now
  - Add 44 to s
  - getMax() should return 44 now
  - Add 0 to s
  - getMax() should still return 44

Break

We get a mandated 10-minute break because our classes are long

Lab Time

Connecting to the class server (server.lawtonsclass.com)

to log out:

exit Ctrl-d

- Download PuTTY if you're on Windows, open the Terminal if you're on a Mac
  - PuTTY: host server.lawtonsclass.com, port 22, SSH, click Open
  - Terminal: ssh username@server.lawtonsclass.com
- Your default username/password is lastname_studentID just like for the class website
  - **You won't see stars** when you type your password. Just type and press Enter.
- Shout out if you're having trouble connecting—we'll move on once all the problems are solved
- **Save your work often**

You don't have to work on the class server

- You just need to turn stuff in on it
- To copy files between the server and your computer, install Cyberduck and use the SFTP protocol on port 22
  - https://cyberduck.io/download/
- Talk to me and I'll be happy to help you get Linux installed on your own computer as a virtual machine/etc.

WinSCP

Quick intro to the shell

- Moving around and seeing where you are
- ~ is short for your home folder
- Editing files with nano (ewww)
- Compiling and running a C++ program

Vim

- The best editor for you to learn at this point in your CSCI career
- Everybody do `vimtutor` if you're not familiar with vim
  - Once you finish, you're free to go. If you don't finish, I encourage you to save your work and complete the tutorial another time.

To be continued...

Use the Assignments tab to see which sections are part of your zyBooks assignment!

(Only the Canvas due dates are correct)

At a minimum, how many test cases should you have for the following code?

```
if (…) {
    …
} else if (…)
    …
} else {
    …
}
```

(a) 1
(b) 2
(c) 3
(d) 4
(e) The limit does not exist

What will take longer as N approaches infinity?

int sum = 0;

1 + 2 + 3 + ... + N

(a) Summing N numbers

(b) Drawing a box of stars of width N

Example of a width 3 box:

N=3

```
* * *
* * *
* * *
```

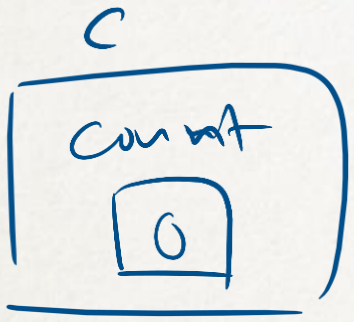
N × N = N^2 *'s

Constructors, initializer lists

Counter *example*

E.g., Wii Bowling keeping track of how many times you messed up and threw the ball in the gutter

Counter c;



init(), inc(), reset(), getCount(), private count

↑ we have to remember to always call init first, which is no fun :(

Constructors are for initialization

- A **constructor** gets called right after an object is born
  - It's a method with the **same name** as the class and no return type. It's a good place to initialize all your vars.
- Let's get rid of our init member function
- Constructors always get called for you, implicitly or explicitly
- **Default constructor**—a constructor that requires no arguments

The default default constructor (not a typo)

- The C++ compiler makes a default constructor for you if you don’t make one yourself
- It calls the default constructors of the member variables in the order that they’re declared

(More on this later)

Initializer lists

Member var names

Example:

```
Blah::Blah(int initialBlorg) :
  blorg(initialBlorg), blerg(42) {
  // body not necessary and can be left blank
}
```

Constructors are called for your member vars **before** you hit the constructor's body. Initializer lists let you call a non-default constructor.

Let's use this on our Counter class!

This is the **only** way to call a constructor on a member "of class type" or to set a const member

Remember the sad face) :

Crushed ICE

- Assume we have a class that looks like

```
class Point {
  public:
    // FIXME Point(int x, int y);
  private:
    int x, y;
};
```

```
Point::Point(int x, int y) :
    x(x), y(y) { }
```

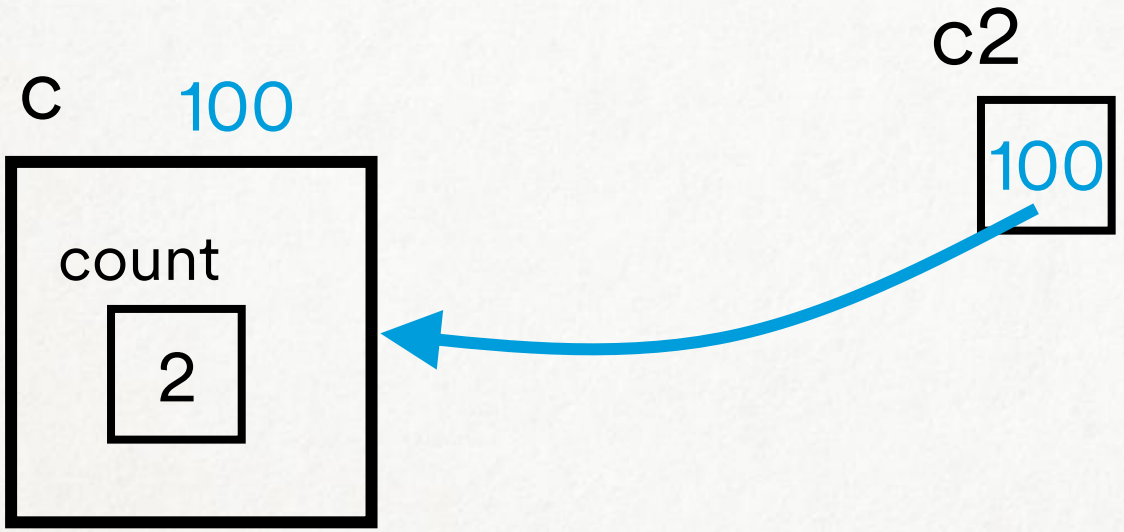
- Using initializer lists, make a constructor for the Point class so that you can do `Point p(3, 5);`

const member functions

- **const** can also be applied to the **end** of member functions!
  - Not to be confused with const reference **parameters** (which we like!)—those guarantee that the method/function won't mess with your variable
- const member functions are when you want to ensure that **you** cannot modify **your own** member variables
  - const member functions are sanity checks for you as the programmer
- getCount() should be const!

Brief intermission: pointers to classes

Know the difference between the * and the ->




```
Counter c;
c.inc();
c.inc();
Counter* c2 = &c;
```

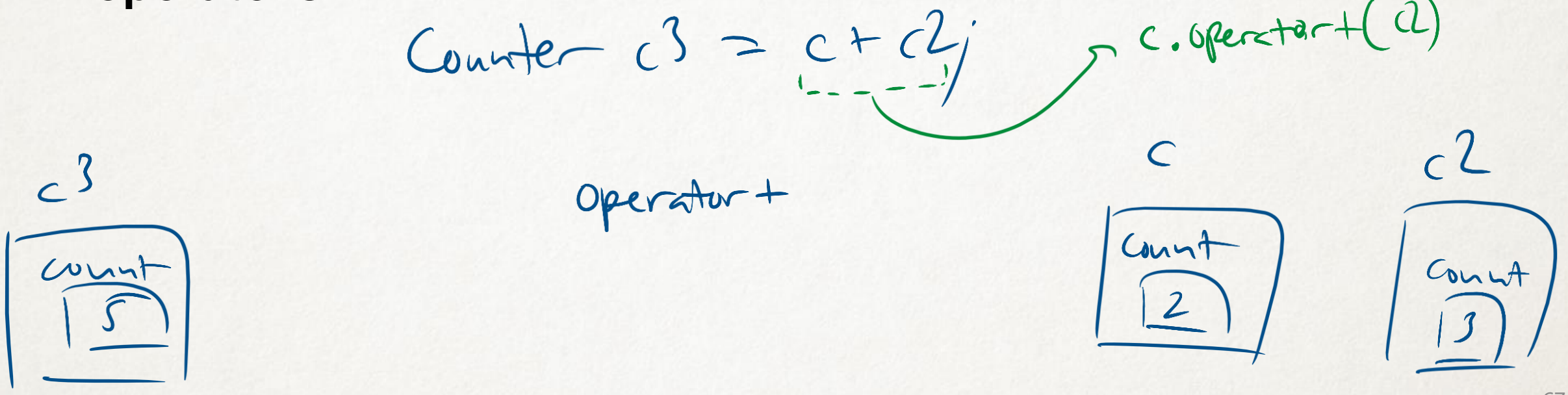
Best practices for classes

- Make all your member variables **private**
  - Write **getters** and **setters** as necessary
- Write constructors that initialize **all** your member variables
- Make as few methods as possible **public**
- Make as many methods as possible **const**
- Pass around (**const**) **references** to save needless copies from getting made
- **Test all public member functions extensively**

Operator Overloading

Let's make a + operator to combine two Counters

- Could make it inside the class or outside the class—let's make this one **inside** the class as a method
- Operators are just functions/methods—their names look like **operatorOP**



*Let's make a * operator to combine two Counters*

- Could make it inside the class or outside the class—let's make this one **outside** the class as just a regular function
- Sometimes the operator needs to be a **friend**, if you need to access to private member vars

Every operator under the sun works similarly

- See https://en.cppreference.com/w/cpp/language/operators and https://en.cppreference.com/w/cpp/language/operator_arithmetic

The output operator

cout << cy;
⇓
operator<<(cout, cy);

```
ostream& operator<<(ostream& os, const YourTypeHere& obj);
```

the type of cout

Give back cout
so you can chain
<< operators

cout
(cout << cy) << endl;
⇓
cout << endl;

This also will work for file streams! (i.e., ofstream)

You can do the opposite with cin/ifstream by changing ostream to istream and << to >>

cin >> cy;